下载APP

13 | 优化方法：更新模型参数的方法

2021-11-08 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远

时长 12:08 大小 11.13M

你好，我是方远。

在上节课中，我们共同了解了前馈网络、导数、梯度、反向传播等概念。但是距离真正完全了解神经网络的学习过程，我们还差一个重要的环节，那就是优化方法。只有搞懂了优化方法，才能做到真的明白反向传播的具体过程。

领资料

今天我们就来学习一下优化方法，为了让你建立更深入的理解，后面我还特意为你准备了一个例子，把这三节课的所有内容串联起来。

用下山路线规划理解优化方法

深度学习，其实包括了三个最重要的核心过程：模型表示、方法评估、优化方法。我们上节课学习的内容，都是为了优化方法做铺垫。

优化方法，指的是一个过程，这个过程的目的就是，寻找模型在所有可能性中达到评估效果指标最好的那一个。我们举个例子，对于函数 f(x)，它包含了一组参数。

这个例子中，优化方法的目的就是**找到能够使得 f(x) 的值达到最小值**对应的权重。换句话说，优化过程就是找到一个状态，这个状态能够让模型的损失函数最小，而这个状态就是**模型的权重**。

常见的优化方法种类非常多，常见的有梯度下降法、牛顿法、拟牛顿法等，涉及的数学知识也更是不可胜数。同样的，PyTorch 也将优化方法进行了封装，我们在实际开发中直接使用即可，节省了大量的时间和劳动。

不过，为了更好地理解深度学习特别是反向传播的过程，我们还是有必要对一些重要的优化方法进行了解。我们这节课要学习的梯度下降法，也是深度学习中使用最为广泛的优化方法。

梯度下降其实很好理解，我给你举一个生活化的例子。假期你跟朋友去爬山，到了山顶之后忽然想上厕所，需要尽快到达半山腰的卫生间，这时候你就需要规划路线，该怎么规划呢？

在不考虑生命危险的情况下，那自然是怎么快怎么走了，能跳崖我们绝不走平路，也就是说：越陡峭的地方，就越有可能快速到达目的地。

所以，我们就有了一个送命方案：每走几步，就改变方向，这个方向就是朝着当前最陡峭的方向，即坡度下降最快的方向行走，并不断重复这个过程。这就是梯度下降的最直观的表示了。

在上节课中我们曾说过：**梯度向量的方向即为函数值增长最快的方向，梯度的反方向则是函数减小最快的方向**。

梯度下降，就是梯度在深度学习中最重要的用途了。下面我们用相对严谨的方式来表述梯度下降。

在一个多维空间中，对于任何一个曲面，我们都能够找到一个跟它相切的超平面。这个超平面上会有无数个方向（想想这是为什么？），但是这所有的方向中，肯定有一个方向是**能够使函数下降最快**的方向，这个方向就是梯度的反方向。每次优化的目标就是沿着这个最快下降的方向进行，就叫做梯度下降。

具体来说，在一个三维空间曲线中，任何一点我们都能找到一个与之相切的平面（更高维则是超平面），这个平面上就会有无穷多个方向，**但是只有一个使曲线函数下降最快的梯度**。再次叨叨一遍：每次优化就沿着梯度的反方向进行，就叫做梯度下降。使什么函数下降最快呢？答案就是损失函数。

这下你应该将几个知识点串联起来了吧：**为了得到最小的损失函数，我们要用梯度下降的方法使其达到最小值**。这两节课的最终目的，就是让你牢牢记住这句话。

我们继续回到刚才的例子。

图中红色的线路，是一个看上去还不错的上厕所的路线。但是我们发现，还有别的路线可选。不过，下山就算是不要命地跑，也得讲究方法。

就比如，步子大小很重要，太大的话你可能就按照上图中的黄色路线跑了，最后跑到了别的山谷中（函数的局部极小值而非整体最小值）或者在接近和远离卫生间的来回震荡过程中，结果可想而知。但是如果步伐太小了，则需要的时间就很久，可能你还没走到目的地，就坚持不住了（蓝色路线）。

在算法中，这个步子的大小，叫做学习率（learning rate）。因为步长的原因，理论上我们是不可能精确地走到目的地的，而是最后在最小值的某一个范围内不断地震荡，也会存在一定的误差，不过这个误差是我们可以接受的。

在实际的开发中，如果损失函数在一段时间内没有什么变化，我们就认为是到达了需要的“最低点”，就可以认为模型已经训练收敛了，从而结束训练。

常见的梯度下降方法

我们搞清楚了梯度下降的原理之后，下面具体来看几种最常用的梯度下降优化方法。

1. 批量梯度下降法（Batch Gradient Descent，BGD）

线性回归模型是我们最常用的函数模型之一。假设对于一个线性回归模型，y 是真实的数据分布函数，$h_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$ 是我们通过模型训练得到的函数，其中θ是 h 的参数，也是我们要求的权值。

损失函数 J(θ) 可以表述为如下公式：

$$\text{cost} = J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^i\right) - y^i\right)^2$$

在这里，m 表示样本数量。既然要想损失函数的值最小，我们就要使用到梯度，还记得我们反复说的“**梯度向量的方向即为函数值增长最快的方向**”么？让损失函数以最快的速度减小，就得用梯度的反方向。

首先我们对 J(θ) 中的θ求偏导数，这样就可以得到每个θ对应的梯度：

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^i\right) - y^i\right) x_j^i$$

得到了每个θ的梯度之后，我们就可以按照下降的方向去更新每个θ，即：

$$\theta_j' = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^i \right) - y^i \right) x_j^i$$

其中α就是我们刚才提到的学习率。更新θ之后，我们就得到了一个更新之后的损失函数，它的值肯定就会更小，那么我们的模型就更加接近于真实的数据分布了。

在上面的公式中，你注意到了 m 这个数了吗？没错，这个方法是当所有的数据都经过了计算之后再整体除以它，即把所有样本的误差做平均。这里我想提醒你，在实际的开发中，往往有百万甚至千万数量级的样本，那这个更新的量就很恐怖了。所以就需要另一个办法，随机梯度下降法。

2. 随机梯度下降（Stochastic Gradient Descent，SGD）

随机梯度下降法的特点是，每计算一个样本之后就要更新一次参数，这样参数更新的频率就变高了。其公式如下：

$$\theta_j' = \theta_j - \alpha \left(h_\theta \left(x^i \right) - y^i \right) x_j^i$$

想想看，每训练一条数据就更新一条参数，会有什么好处呢？对，有的时候，我们只需要训练集中的一部分数据，就可以实现接近于使用全部数据训练的效果，训练速度也大大提升。

然而，鱼和熊掌不可兼得，SGD 虽然快，也会存在一些问题。就比如，训练数据中肯定会存在一些错误样本或者噪声数据，那么在一次用到该数据的迭代中，优化的方向肯定不是朝着最理想的方向前进的，也就会导致训练效果（比如准确率）的下降。最极端的情况下，就会导致模型无法得到全局最优，而是陷入到局部最优。

世间安得两全法，有的时候舍弃一些东西，我们才能获得想要的。**随机梯度下降方法选择了用损失很小的一部分精确度和增加一定数量的迭代次数为代价，换取了最终总体的优化效率的提高**。

当然这个过程中增加的迭代次数，还是要远远小于样本的数量的。

那如果想尽可能折衷地去协调速度和效果，该怎么办呢？我们很自然就会想到，每次不用全部的数据，也不只用一条数据，而是用“一些”数据，这就是接下来我们要说的小批量梯度下降。

3. 小批量梯度下降（Mini-Batch Gradient Descent, MBGD）

Mini-batch 的方法是目前主流使用最多的一种方式，它每次使用一个固定数量的数据进行优化。

这个固定数量，我们称它为 batch size。batch size 较为常见的数量一般是 2 的 n 次方，比如 32、128、512 等，越小的 batch size 对应的更新速度就越快，反之则越慢，但是更新速度慢就不容易陷入局部最优。

其实具体的数值设成为多少，也需要根据项目的不同特点，采用经验或不断尝试的方法去进行设置，比如图像任务 batch size 我们倾向于设置得稍微小一点，NLP 任务则可以适当的大一些。

基于随机梯度下降法，人们又提出了包括 momentum、nesterov momentum 等方法，这部分知识同学们有兴趣点击这里可以自行查阅。

一个简单的抽象例子

我们通过三节课（第 11 到 13 节课），分别学习了损失函数、反向传播和优化方法（梯度下降）的概念。这三个概念也是深度学习中最为重要的内容，其核心意义在于能够让模型真正做到不断学习和完善自己的表现。

那么接下来我们将通过一个简单的抽样例子把三节课的内容汇总起来。需要注意的是，下面的例子不是一个能够运行的例子，而是旨在让我们更加明确一个最基本的 PyTorch 训练过程都需要哪些步骤，你可以当这是一次军训。有了这个演示例子，以后我们上战场，也就是实现真正可用的例子也会事半功倍。

在一个模型中，我们要设置如下几个内容：

模型定义。

损失函数定义。

优化器定义。

通过下面的代码，我们来一块了解一下，上面三个内容在实际开发中应该怎么组合。当然，这个代码是一个抽象版本，目的是帮你快速领会思路。具体的代码填充，还是要根据实际项目来修改。

复制代码

```
import LeNet #假定我们使用的模型叫做LeNet，首先导入模型的定义类
import torch.optim as optim #引入PyTorch自带的可选优化函数
...
net = LeNet() #声明一个LeNet的实例
criterion = nn.CrossEntropyLoss() #声明模型的损失函数，使用的是交叉熵损失函数
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
# 声明优化函数，我们使用的就是之前提到的SGD，优化的参数就是LeNet内部的参数，lr即为之前提到的

#下面开始训练
for epoch in range(30): #设置要在全部数据上训练的次数

    for i, data in enumerate(traindata):
        #data就是我们获取的一个batch size大小的数据

        inputs, labels = data #分别得到输入的数据及其对应的类别结果
        # 首先要通过zero_grad()函数把梯度清零，不然PyTorch每次计算梯度会累加，不清零的话
        optimizer.zero_grad()
        # 获得模型的输出结果，也即是当前模型学到的效果
        outputs = net(inputs)
        # 获得输出结果和数据真正类别的损失函数
        loss = criterion(outputs, labels)
        # 算完loss之后进行反向梯度传播，这个过程之后梯度会记录在变量中
        loss.backward()
        # 用计算的梯度去做优化
        optimizer.step()
...
```

这个抽象框架是不是非常清晰？我们先设置好模型、损失函数和优化函数。然后针对每一批（batch）数据，求得输出结果，接着计算损失函数值，再把这个值进行反向传播，并利用优化函数进行优化。

别看这个过程非常简单，但它是深度学习最根本、最关键的过程了，也是我们通过三节课学习到的最核心内容了。

总结

这节课，我们学习了优化方法以及梯度下降法，并通过一个例子将损失函数、反向传播、梯度下降做了串联。至此，我们就能够在给定一个模型的情况下，训练属于我们自己的深度学习模型了，恭喜你耐心看完。

当你想不起来梯度下降原理的时候，不妨回顾一下我们下山路线规划的例子。我们的目标就是设置合理的学习率（步伐），尽可能接近咱们的目的地（达到较理想的拟合效果）。用严谨点的表达说，就是正文里咱们反复强调的：**为了得到最小的损失函数，我们要用梯度下降的方法使其达到最小值。**

这里我再带你回顾一下这节课的要点：

> 模型之所以使用梯度下降，其实是通过优化方法不断的去修正模型和真实数据的拟合差距。
>
> 常用的三种梯度方法包括批量、随机和小批量，一般来说我们更多采用**小批量梯度下降。**
>
> 最后我们通过一个抽象的框架，汇总了训练一个模型所需要的几个关键内容，如损失函数、优化函数等，这部分内容是深度学习最关键的过程，建议你重点关注。

每课一练

batch size 越大越好吗？

欢迎你在留言区记录你的疑问或收获，也推荐你把这节课分享给更多的同事、朋友。

我是方远，我们下节课见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 计算梯度：网络的前向与反向传播

下一篇 加餐 | 机器学习其实就那么几件事

精选留言 (7)

写留言

vcjmhg
2021-11-09

不是
1. batch_size越大显存占用会越多，可能会造成内存溢出问题，此外由于一次读取太多的样本，可能会造成迭代速度慢的问题。
2. batch_size较大容易使模型收敛在局部最优点
3. 此外过大的batch_size的可能会导致模型泛化能力较差的问题

展开

作者回复: 👍👍👍^^，厉害。

1

clee
2021-11-08

你好，最近3节课是理论偏多，实际动手部分偏少，那这部分内容应该怎么实际动手练习才能和理论想结合的更好，这方面有什么建议呢？

展开

作者回复: 你好，clee，感谢你的留言。
先恭喜你坚持到这了^^。在14节课里会把前面的知识串起来，讲讲如何自己构建一个模型，然后训练它。
在后续的实战篇里，会讲解在工业中一些CV与NLP的应用，在实战篇里都能看到这些理论知识的影子。

1

和你一起搬砖的胡大爷
2021-11-17

老师，吴恩达老师的ml课上说不管什么gd都没法确定打到全局最优，在高维数据下几乎必然只能到局部最优，所以初始化的技巧就比较重要。
老师您说的就无法得到全局最优我不大理解

作者回复: hello, 其实这跟Andrew Ng大神的说法不冲突的，咱们在这里讨论的是一种相对简单（特征没有那么多）的情况，形象化的比喻的话，还是下山的问题，高维数据相当于不是一座山，而是一片片山脉，模型在寻找最优的过程中因为步长的原因，极大概率就会进入到局部最优。这也就是为什么有时候会加入一个momentum的原因，让模型再冲出去到别的山谷试试。

马克图布

2021-11-12

文章里说：

> 越小的 batch size 对应的更新速度就越快，反之则越慢，但是更新速度慢就不容易陷入局部最优。

...

展开

作者回复: hello。
batch size太小的话，那么每个batch之间的差异就会很大，迭代的时候梯度震荡就会严重，不利于收敛。

batch size越大，那么batch之间的差异越小，梯度震荡小，利于模型收敛。

但是凡事有个限度，如果batch size太大了，训练过程就会一直沿着一个方向走，从而陷入局部最优。

这也就是为什么我们要不断的尝试一个相对合理的mini batch

官
2021-11-08

并不是，batchsize越大更新越慢，而且对显存要求变高，在配置较低的机器会出现显存不足的问题

作者回复: ^^，👍👍。是的，除此之外，较大的batch_size容易使模型收敛在局部最优点，特别小则容易受噪声影响。

IUniverse
2021-11-08

batchsize越大，更新模型的也会越慢。同时太大了也会爆显存

作者回复: 你好，IUniverse，谢谢你的留言。
是的，
除此之外，较大的batch_size容易使模型收敛在局部最优点，特别小则容易受噪声影响。^^

tom

2021-11-08

batch size过大，有可能会爆显存。。。

展开

作者回复: 你好，tom，谢谢你的留言。

是的，batch size太大显存会爆掉。

除此之外，较大的batch_size容易使模型收敛在局部最优点，特别小则容易受噪声影响。^^